The Structural Bioinformatics Library: modeling in biomolecular science and beyond

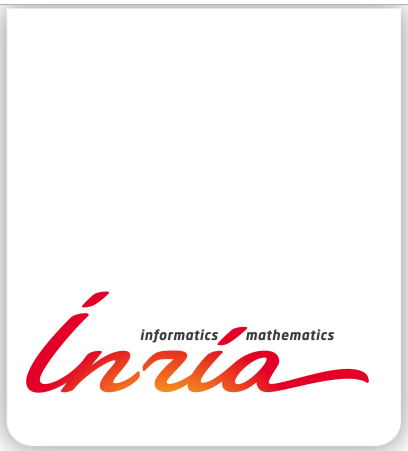
Frédéric Cazals, Tom Dreyfus

The Structural Bioinformatics Library: modeling in biomolecular science and beyond

F. Cazals and T. Dreyfus

RESEARCH REPORT
N° 8957
September 2016
Project-Team ABS

ISSN 0249-6399 ISRN INRIA/RR--8957--FR+ENG



The Structural Bioinformatics Library: modeling in biomolecular science and beyond

F. Cazals and T. Dreyfus

Project-Team ABS

Research Report n° 8957 — September 2016 — 22 pages

Abstract: **Motivation:** Software in structural bioinformatics has mainly been application driven. To favor practitioners seeking off-the-shelf applications, but also developers seeking advanced building blocks to develop novel applications, we undertook the design of the Structural Bioinformatics Library (`SBL`, `http://sbl.inria.fr`), a generic C++/python cross-platform software library targeting complex problems in structural bioinformatics. Its tenet is based on a modular design offering a rich and versatile framework allowing the development of novel applications requiring well specified complex operations, without compromising robustness and performances.
Results: The `SBL` involves four software components (1-4 thereafter). For end-users, the `SBL` provides ready to use, state-of-the-art *(1) applications* to handle molecular models defined by unions of balls, to deal with molecular flexibility, to model macro-molecular assemblies. These applications can also be combined to tackle integrated analysis problems. For developers, the `SBL` provides a broad C++ toolbox with modular design, involving core *(2) algorithms*, *(3) biophysical models*, and *(4) modules*, the latter being especially suited to develop novel applications. The `SBL` comes with a thorough documentation consisting of user and reference manuals, and a bugzilla platform to handle community feedback.
Availability: The `SBL` is available from `http://sbl.inria.fr`

Key-words: Structural bioinformatics, Scientific software, Software libraries

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

La *Structural Bioinformatics Library*: modélisation des systèmes biomolécules et au-delà

Résumé : **Motivation:** Le logiciel pour la bioinformatique a essentiellement été motivé par les applications. De façon à satisfaire les utilisateurs cherchant des applications clef en main, mais aussi les developpeurs ayant besoin de briques logicielles afin de créer de nouvelles applications, nous avons développé la Structural Bioinformatics Library (`SBL`, `http://sbl.inria.fr`), une librairie générique C++/python *cross-platform*. Elle offre un design modulaire et un environnement riche en briques algorithmiques complexes, sans compromis sur la robustesse et les performances.

Resultats: La `SBL` offre quatre compartiments logiciels. Pour les utilisateurs en fin de chaîne, elle propose des *(1) applications* permettant de manipuler des modèles moléculaires définis par des unions de boules, d'explorer la flexibilité des molécules, et de modéliser de gros assemblages. De plus, son design rend aisé la combinaison de ces applications, permettant de mener à bien des analyses intégrées. Pour les développeurs, la `SBL` propose une boite à outil C++ à large spectre, au design modulaire, avec *(2) algorithmes*, *(3) modèles biophysiques*, et *(4) modules*, ces derniers rendant trivial le développement de nouvelles applications. Par ailleurs, une documentation exhaustive est proposée (user et référence manuels), ainsi qu'une plate-forme bugzilla pour la gestion des bogues.

Accès:: rendez-vous sur `http://sbl.inria.fr`

Mots-clés : Bioinformatique structurale, logiciel scientifique, librairie scientifique

Contents

1 Introduction

1.1 Software for computational structural biology

Computational structural biology is concerned with the investigation of the relationship between the structure and dynamics of biomolecules on the one hand, and their function on the other hand. The questions in this realm are especially challenging due to the dynamic nature of the phenomena studied, the time-scales involved, and the size of systems, which vary from small molecules or peptides to large assemblies involving hundreds of macro-molecules. Another source of complication is the hybrid nature of the relevant data, as insights on complex systems typically mix experiments (X ray crystallography, NMR, electron microscopy, mass spectrometry) and modeling. This state of affairs is such that a very large number of software applications and libraries have been developed in structural bioinformatics, with more than 350 classified at `http://www.ks.uiuc.edu/Development/biosoftdb/biosoft.cgi`. In this offer, several large software tools stand out. Platforms such as `MODELLER` ([1] and `http://salilab.org/modeller`), `Rosetta` ([2] and `https://www.rosettacommons.org/software`), or the Integrative Modeling Platform ([3] and `https://integrativemodeling.org`) enabled notable scientific advances in biophysics and computational biology. Docking algorithms at large are invaluable to predict the structure of complexes ([4] and `https://en.wikipedia.org/wiki/Docking_(molecular)`). Force fields and the associated simulation platforms, including `CHARMM` ([5] and `https://www.charmm.org`), `AMBER` ([6] and `http://ambermd.org`), `NAMD` ([7] and `http://www.ks.uiuc.edu/Research/namd`), `GROMACS` ([8] and `http://www.gromacs.org`), or the Cambridge suite software suite for energy landscapes ([9] and `http://www-wales.ch.cam.ac.uk/software.html`), proved invaluable to study the dynamics of (bio-)molecules. Likewise, molecular visualization programs such as `VMD` ([10] and `http://www.ks.uiuc.edu/Research/vmd`), `Pymol` ([11] and `https://www.pymol.org/`), `Chimera` ([12] and `https://www.cgl.ucsf.edu/chimera/`), or `SAMSON` (`http://www.samson-connect.net`) leverage the analysis of simulation results. Finally, BioPython ([13] `http://biopython.org/`) provides several tools in the realm of computational biology–even if its focus is not on structure.

1.2 Software design

The aforementioned software environments are major scientific achievements. However, they usually do not clearly disentangle the end-user applications solving well specified biophysical problems, and the underlying low level algorithmic classes. This in turn hinders the design of software solutions combining complementary building blocks from various sources, as no unified design principle is common to most libraries.

To change the state of affairs, we undertook the design and development of the Structural Bioinformatics Library (`http://sbl.inria.fr`), a highly modular software library matching the best standards in two directions. On the one hand, similarly to the previously cited environments, the `SBL` offers end-user applications with state-of-the-art performances. On the other hand, these applications are based on low-level algorithmic classes whose design matches that of comprehensive C++ libraries such as the `Computational Geometry Algorithms Library` (CGAL, `http://www.cgal.org`) or the boost C++ libraries (`http://www.boost.org/`). This unique combination makes it possible to develop novel challenging applications, by reusing highly parameterized software components. In the sequel, we present the main software components geared towards end-users and developers.

2 The SBL for end-users

2.1 Rationale

From the structural bioinformatics standpoint, the SBL aims to provide reference methods solving specific problems. The methods are grouped by-so called applications, themselves ascribed to categories depending on the models manipulated (for the *Applications page* of the web site, see http://sbl.inria.fr/applications/; for the *Applications page* from the on-line documentation, see http://sbl.inria.fr/doc/group__sbl-applications.html; see also Fig. 1). The groups of applications are:

• *Space filling models:* applications dealing with molecular models defined by unions of balls, e.g. van der Waals or solvent accessible models [14]. Prototypical applications provide interface modeling (Figs. 1(A) and 2, [15, 16, 17]), the calculation of molecular surfaces and volumes [18, 19], or the design of coarse grain models [20].

• *Conformational analysis:* applications dealing with molecular flexibility [9, 21]. Reference methods are provided to sample potential energy landscapes [22] (Fig. 1(B)), to analyze and compare ensembles of conformations and sampled energy landscapes [23, 24]. Future additions will focus on novel methods to compute thermodynamic and kinetic properties of bio-molecular systems, with a focus on Monte Carlo based methods (in particular Wang-Landau type algorithms to compute densities of states[25]), as well as methods exploiting the formalism of energy landscapes[9].

• *Large assemblies:* applications dealing with macro-molecular assemblies involving from tens to hundreds of subunits [26]. Particularly noticeable is the application inferring contacts between subunits of a large assembly from native mass spectrometry data [27], as it doubles performances of competing methods [28] (Fig. 1(C)). Future additions will focus on methods to assess [29, 30] models stemming from reconstruction by data integration [31, 3].

• *Integrated analysis:* Applications combining several ingredients from the previous groups. An application is provided to estimate binding affinities of protein complexes, with state-of-the-art results [32] on the reference structure-affinity benchmark [33] (Fig. 1(D)).

To ease the reproducibility of computer experiments, the SBL also proposes specific data management and analysis methods:
• *Data Management:* applications easing data management in large scale computer experiments.

• *Data Analysis:* applications providing novel data analysis - statistical analysis tools.

Using selected of these applications is especially easy, since an auto-installation script (available at http://sbl.inria.fr/applications/) makes them directly available under VMD and Pymol. For example, under VMD, the interface analysis of Fig. 2 are readily reproduced using the applications available from the menu Extensions > SBL > Intervor > { sbl_intervor_ABW_atomic, sbl_intervor_IGAgW_atomic }. More generally, the user manual of each application provides a detailed documentation, with in particular input files, output files, and command lines with full lists of options.

2.2 Key features

From the software design standpoint, all applications meet the best practices of modern large scale software libraries:

• *Versatility.* In the SBL, an application is a set of programs characterized by inputs and outputs obeying the same semantics. We illustrate this on the problem of modeling macro-molecular *interfaces* (Fig. 2). Intuitively, an interface between two partners separates these partners, and should therefore be oblivious to their nature (proteins, nucleic acids, metabolites, etc) or resolution (at interface may be defined at any resolution). Also, the notion of interface should cover cases where the partners have a specific structure. For example, in modeling an antibody - antigen complex, the antibody fragment antigen binding domain (FAB) may be decomposed hierarchically into domains from the heavy and light chains, or into so-called complementarity determining regions and framework regions. The SBL handles all such cases at once, making it possible to study a given object at different levels of details, or to study objects with the same semantic using the same concepts. The case of interfaces is explained in the documentation at `http://sbl.inria.fr/doc/Space_filling_model_interface-user-manual.html`. More generally, the derivation of programs within a given application is explained in section 3.

• *Documentation.* Each application comes with a detailed user manual, stating the main goals, defining the pre-requisites, and specifying the inputs and outputs. Moreover, example runs of the programs are provided, together with the associated input and output files.

• *Specifications and workflows.* All applications come with a sharp specification of pre-requisites, inputs and outputs. Most importantly, each application comes with a *workflow*, namely a graph specifying the sequence of operations, each undertaken by a so-called module (Fig. 3 and section 3). The workflow of an application is automatically generated by this application, which guarantees coherence with the source code and the claimed functionalities. It is instrumental to view the application at a glance, and to design data processing pipelines.

• *Perennial data formats.* Programs from the SBL generate results in perennial data format, in particular XML. The hierarchical structure of XML, in conjunction with the XPath query language, allows automating data storage, parsing and retrieval, so that upon running calculations with applications, statistical analysis and plots are a handful of Python lines away. This also allows a trivial coupling with data analysis modules provided in `Scikit-learn` (`http://scikit-learn.org/stable/`) or `R` (`https://www.r-project.org/`).

• *Reproducibility.* A critical point in bioinformatics is the ability to reproduce results [34]. To this end, the SBL provides tools to configure, execute and repeat large scale computer experiments. Automating computer experiments is of paramount importance in particular when randomness is at play, e.g. in cross validation experiments, Monte Carlo based exploration of conformational spaces, or non convex optimization.

• *Visualization.* Whenever relevant, applications from the SBL are coupled to visualization plugins for `VMD` and `PyMol`. In fact, automatic installation scripts are provided for these two environments (`http://sbl.inria.fr/applications/`), so that all executables from the SBL are two clicks away from the `VMD` and `PyMol` menus.

• *Ability to handle complex multi-scale biophysical problems.* Broadly speaking, the modeling problems faced in structural bioinformatics may be ascribed to three categories. Atomic resolution models aim at unveiling fine structural and dynamical properties of molecules or (small) complexes. Such models typically require modeling the flexibility of molecules, to obtains thermodynamic and/or kinetic descriptors [9, 21]. Coarse-grain models aim at unveiling the structure of larger systems, by giving access to the overall shape, and possibly to pairwise contacts within an assembly. Such models typically rely on various experimental data such as mass spectrometry, cryo electron microscopy or SAXS data [31, 3]. Finally, hybrid models aim at approaching atomic resolution models, by combining low resolution models of an assembly with high resolution structures of subunits [35].

Most of the problems in these three realms can be tackled using two types of representations, based on union of balls on the one hand (van der Walls, solvent accessible models, coarse grain models), and coordinate based representations on the other hand (internal coordinates or Cartesian coordinates). The SBL provides such formats, with easy conversion between them–from a geometric standpoint, so as to foster inter-connections between atomic resolution, low resolution, and hybrid models.

• *Robustness and performances.* Structural data are geometric in nature, which requires handling possibly degenerate situations so as to guarantee robustness and accuracy of results. To see why, consider the seemingly simple problem of computing the volume of a molecule, so as e.g. to assess the atomic packing properties. This calculation is generally undertaken using floating point numbers, which typically yields errors of 5-10% [18, 19]. The SBL resorts to numerically certified operations instead, which warrants robust programs and accurate results. This entails using appropriate number types (with arbitrary precision, interval number types) when appropriate. Likewise, the algorithms have been optimized, so as to guarantee that their complexity scales ideally as a function of the data size.

• *Modularity and extensibility.* Addition of new applications is especially easy, due to the coherence between the representations just discussed, and the highly modular structure of the library as detailed below.

• *Interoperability.* Applications are inter-operable in three respects. First, standard file formats are used as input. The PDB format is generally used for molecular conformations. But all algorithms are templated by so-called loaders, which makes it especially easy to import data from any (custom) format. Second, results are stored in XML files, to ease the integration of applications within bioinformatics pipelines. Third, on the programming side, the source code can be directly embedded into external applications, as detailed below.

3 The SBL for developers

3.1 Rationale

In the perspective of software development, the design of the SBL has been guided by two main principles. The first one is to provide a broad C++/python toolbox, fostering code re-usability and the development of complex applications with clear workflows. In a nutshell, C++ is used whenever generic classes are in order–see our description of template based design in section 3.2.
The second is to offer a modular design fostering the integration and exchange of core building blocks, be they from computer science or biophysics. These principles aim at accommodating the complexity of software in structural bioinformatics, which inherently mixes two difficulties. The first one relates to the variety of models used in biophysics at large, namely models from physics (e.g., potential and free energy models), chemistry (e.g., force fields), and biology (e.g., sequence - structure information, functional annotations, etc). The second one pertains to the variety of concepts and algorithms essentially covering the whole spectrum of computer science and applied mathematics, including combinatorial algorithms and data structures (e.g., sorting and searching methods, graph algorithms), numerics (e.g, number types, optimization methods), geometric and topological methods (e.g., Delaunay and Voronoi representations, spatial search data structures), data analysis.

Through its modular design, the SBL fosters the integration of software components in these two realms, which also warrants code re-usability and the development of novel applications.

3.2 Structure of the SBL

To further describe the structure of the SBL (Fig. 4), some familiarity with generic software in general and C++ in particular is essential [36]. In C++, a *template class* is a class accommodating different data types providing the same pre-requisites, upon so-called *instantiation*. For example, a simple list may be a list of integers, floats, proteins, complexes, etc. This mechanism avoids rewriting basic classes for specific types. The requirements of a template class are *concepts*, a concept being a set of types and methods. A *model* for a concept is a class implementing these types and methods. Within the SBL, we distinguish two categories of template parameters: (i) the internal types, namely concepts for which default models are provided, and (ii) the external types, namely concepts depending on the software component developed. This classification also simplifies the instantiation of template classes, since external types are grouped within *traits classes*. (In particular, this is always the case for the so-called modules, see below.)

This said, the structure of the SBL is as follows:

- SBL-APPLICATIONS: applications solving specific applied problems, as described in section 2.

- SBL-CORE: C++ classes providing (advanced) algorithms, ascribed to four tiers: CADS: Combinatorial Algorithms and Data Structures, GT: Computational geometry and computational topology, CSB: Computational Structural Biology, IO: Input / Output. Selected classes are templated by traits classes specifying C++ concepts, namely requirements in terms of functionalities.

- SBL-MODELS: C++ *models* coding physical and/or chemical and/or biological knowledge. These classes typically match the C++ concepts required to instantiate those classes from SBL-CORE which are templated.

- SBL-MODULES: C++ classes instantiating classes from SBL-CORE with specific biophysical models from SBL-MODELS. A module may be seen as a black box transforming an input into an output. With modules, an application workflow is merely a graph of interconnected modules, whose automatic traversal generates the application workflow.A module is provided for a package in SBL-CORE when the corresponding algorithms are key steps of an application from SBL-APPLICATIONS. There are currently 15 modules scattered over 13 SBL-CORE packages, as listed at http://sbl.inria.fr/doc/group__sbl-core-modules.html.

3.3 Development of novel programs and applications

Using the structure just discussed, several types of developments can be undertaken.

Recall that an application is a set of programs characterized by inputs and outputs obeying the same semantics. We handle such invariants using generic software design techniques in C++. More precisely, each application is developed using generic concepts resorting to template parameters. For an existing application, a novel program may be obtained by replacing a template parameter by another one sharing the prescribed specifications. For example, given the C++ skeleton of our application dissecting macro-molecular interfaces (Fig. 2), deriving a novel program taking into account specific features of antibodies merely requires replacing the class labeling the atoms of the two partners by a class which in addition ascribes the atoms of the IG to its heavy and light chains. Phrased differently, deriving the new program requires changing one type specification and recompiling.

Novel applications may be developed in two ways. The easiest route consists of combining (pooling) existing modules, as exemplified in the supplementary section 6.. A more challenging route, for experienced C++ programmers, consists in instantiating classes from `SBL-CORE` (existing ones or novel ones) with models from `SBL-MODELS` (existing ones or novel ones).

3.4 Low level developments

Basic algorithms provided in `SBL-CORE` underlie essentially all complex data processing tasks, which we illustrate with three examples. In the **Space Filling Models** group of applications, most of the software components rely on Voronoi diagrams and so-called α-shapes [37], which provide parameter free models of macro-molecules structures and their complexes. Such calculations are notoriously difficult from the combinatorial and numerical standpoint, and our solution relies on a mix of packages from `CGAL` and developed in the `CADS` tier of `SBL-CORE`. For applications dealing with **Conformational Analysis**, a critical operation to deal with large ensembles of conformations is to perform so-called nearest neighbor queries [38], to report e.g. the conformation in a set most similar to a query conformation. Depending on the metric used, these operations require advanced algorithms in metric spaces, which are provided again within `CADS` tier of `SBL-CORE`. Finally, in the **Large assemblies** group, a key problem is to accommodate uncertainties on models. A fundamental tool to assess stable geometric and topological features of such models is topological persistence [39], and again, the `SBL` provides several key algorithms in this realm, e.g. used in the mode seeking based clustering algorithm [40].

It should also be stressed that the modular structure of the `SBL` makes it simple to replace one building block by another one, provided coherent specifications.

3.5 License, releases, distribution

The `SBL` is released under a proprietary open source license, see `http://sbl.inria.fr/license/`. In a nutshell, academic users can use and modify the code at their discretion, for research purposes – commercial exploitation of the `SBL` requires a specific license. Contributions from the outside are also welcome, provided that they comply with the development philosophy. One such example is the `Apurva` package (`http://sbl.inria.fr/doc/group__Apurva-package.html`), which provides reference methods to perform structural alignments [41].

The source code is distributed from `http://sbl.inria.fr`, using tarballs and a git repository. Full support via a Bugzilla interface is provided for linux and MacOS. For windows, individual executables were compiled and tested, but full support is not provided since scripts and plugins were not thoroughly tested.

Applications can be directly downloaded and installed from `http://sbl.inria.fr/applications/`, and the whole documentation is available at
`http://sbl.inria.fr/documentation-and-tutorials/`.

4 Discussion

4.1 High quality applications

Having presented the structure of the `SBL`, we argue that the high quality applications provided rely on the combination of two features, namely a careful specification of the mathematical properties of the algorithms used, and an advanced C++ design which makes it possible to couple elaborate building blocks. We provide one illustration for each group of applications.

Space filling models. Consider the application computing molecular surfaces and volumes, see `http://sbl.inria.fr/doc/Space_filling_model_surface_volume-user-manual.html`. Classical contenders typically resort to `float` or `double` number types to do so, typically yielding errors measured in percents [19], a fact inherent to the rounding of floating point numbers. Our application provides certified results, a property stemming from the instantiation of low-level algorithms from `SBL-CORE` with number types equipped with guarantees (types with arbitrary precision, interval number types).

Conformational analysis. Consider the application exploring potential energy landscapes (PEL) of biomolecular systems, see `http://sbl.inria.fr/doc/Landscape_explorer-user-manual.html`. For (models of) biomolecules, PEL typically involve a very large number of local minima [22, 24]. The efficient exploration of such landscapes requires efficient algorithmic components (random sampling, exploration of high dimensional space, location of nearest neighbors in metric spaces), coupled with the biophysical models (potential energy) scrutinized. In the `SBL`, deriving new exploration programs is especially easy, since one merely needs to instantiate the required low-level algorithms from `SBL-CORE` with biophysical models form `SBL-MODELS`.

Large assemblies. A key difficulty in modeling large assemblies is to harness the uncertainties inherent to coarse-grain models. In the `SBL`, this is done by putting the emphasis on the quality of geometric approximations, and on the enumeration of solutions achieving optimal scores. For example, for coarse grain modeling, see `http://sbl.inria.fr/doc/Space_filling_model_coarse_graining-user-manual.html`, the `SBL` provides the only algorithms producing models with volumetric approximation guarantees, a fact stemming from insights on the NP-complete problems solved. In a different realm, in inferring pairwise contacts for native mass spectrometry data, see `http://sbl.inria.fr/doc/Connectivity_inference-user-manual.html`, our connectivity inference methods enumerate all optimal solutions, which avoids arbitrariness in the solutions exploited.

Integrated analysis. Consider the problem of estimating binding affinities from structural data, see `http://sbl.inria.fr/doc/Binding_affinity_prediction-user-manual.html`. By combining several constructions provided by programs from the space filling model group, the application provided delivers state-of-the-art affinity predictions, with accuracy typically better than 1.4 kcal/mol [32].

4.2 Towards more integration and further applications

The architecture just discussed can be beneficial to design involved applications in molecular science and beyond.

In molecular science, several of our building blocks may be combined to develop novel hybrid and/or integrated models. As an illustrative example, one may consider the computation of absolute binding free energies via accurate thermodynamic calculations [42, 43]. While such calculations are usually based on ab initio calculations using force fields, they could be hybridized with recently designed knowledge based parameters which were instrumental to obtain estimates of unprecedented accuracy [44, 32]. The fact that the `SBL` provides both classes of methods is a significant asset to develop such models. Another example is the design and the assessment of models in the context of reconstruction by data integration [31], where the coupling of improved exploration functions for non convex functionals [22] and delicate geometric and graph theoretical assessment methods [29, 30] may yield improved models.

Beyond molecular science, our low-level algorithms can naturally be used in a variety of contexts. A striking illustration is provided by geometric models based on union of balls, as such models are also encountered in material sciences (e.g. to represent materials and study their

properties), in geometric modeling (e.g. to represent shapes via the medial axis transform), in biomedicine (e.g. to represent approximations of discretized organs), in computer graphics (e.g. to define hierarchies of bounding volumes used in collision checking), or even in astrophysics (e.g. to study the topology of galaxies using α-shapes). In fact, our previous software `Vorlume`, computing surfaces and volumes of unions of balls `Vorlume`[19] was used in all these contexts. With its generic design (section 2.2), the repackaged application `Space_filling_model_surface_volume` goes even further and makes it trivial to accommodate models based on union of balls whose semantics is arbitrary.

5 Outlook

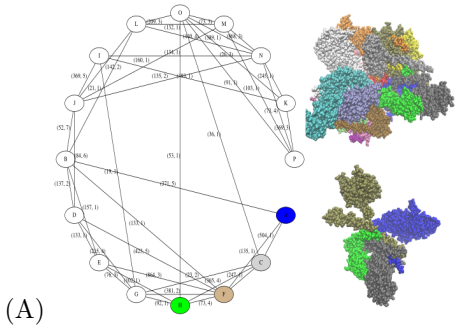
Computer programming may be considered as an art, as a science, or both. Following Donald Knuth in his Turing award lecture [45], ≪ *Science is knowledge that we understand so well that we can teach it to a computer; and if we don't fully understand something, it is an art to deal with it.* ≫. Later in his lecture, Knuth also mentions that a *good program* should (i) work correctly, (ii) be easy to change, (iii) allow graceful interaction with its users, and (iv) be efficient given the computer resources at hand.

The development of software tools for structural bioinformatics in general and within the `SBL` in particular may be considered in light of these thoughts. The correctness is certainly critical, and calls for a careful specification of the inputs and outputs of a program, a tenet for every single application of the `SBL`. Correctness issues were actually considered down to the most exquisite levels, as all algorithms are parameterized by number types which may be adapted to match specific requirements. The ability to easily change a program to accommodate a new setting clearly calls for a versatile design. Advanced software design techniques relying on C++ templates offer such a flexibility, allowing a new program to be compiled by merely changing a type definition. Moreover, for end-users not versed into programming, a vast array of ready to use applications are provided. The graceful interaction with users stems from perennial data formats on the one hand, and the coupling with classical visualization systems on the other hand. Finally, the efficacy of applications is ensured by resorting to low level algorithms providing the best complexities, be they from the `SBL` itself or from external specialized libraries.

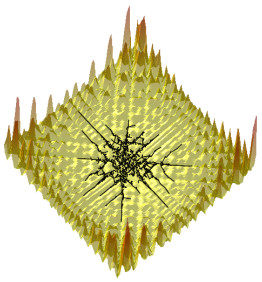
Summarizing, the `SBL` matches the four aforementioned requirements both for end-users and developers. In fact, it is to the best of our knowledge the first library doing so in structural bioinformatics. We anticipate that this unique design will prove invaluable, reducing the development of complex applications to a few days, as opposed to months starting from scratch. Naturally, we also anticipate that in combining insights obtained from complementary advanced modeling techniques such as the ones offered, stimulating views on complex bio-molecular systems will be obtained, and that these will in turn raise novel questions and hypothesis.

Our ambition for the near future is twofold. First, we shall expand the functionalities, both in terms of off-the shelf applications and low level algorithmic classes. Second, we shall welcome and integrate contributions from the community, provided that these comply with the development philosophy.

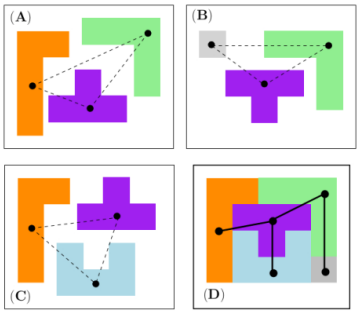
Figure 1 Example end-user applications, see `http://sbl.inria.fr/` > Applications. **(A)** Group **Space filling models**: `Space_filling_model_interface_finder`. A package to seek interfaces of molecular assemblies. A graph is reported, with one node per subunit (chain or domain in a chain), and one edge per interface between any two subunits in contact. Each edge is decorated by the number of interface atoms and the number of patches. Each such interface can be studied in much mode detail (Fig. 2). **(B)** Group **Conformational analysis**: `Landscape_explorer`. A package providing Monte Carlo based algorithms (basin hopping, rapidly exploring random trees, hybrid algorithms) to explore potential energy landscapes of molecular systems. Shown here is a rapidly exploring random tree crawling along the valleys of a 2D mathematical landscape. **(C)** Group **Large assemblies**: `Connectivity_inference`. A package to perform connectivity inference between subunits of large assemblies, from native mass spectrometry data. Given a set of oligomers (one oligomer is one list of subunits, three of them are shown as colored polygons in subpanels (A,B,C)), pairwise interfaces between these subunits are inferred (subpanel D). **(D)** Group **Integrated analysis**: `Binding_affinity_predictions`. A package to perform binding affinity predictions from structures of partners and their complex. For high resolution structures, an accuracy on ΔG_d circa 1 kcal/mol is achieved.



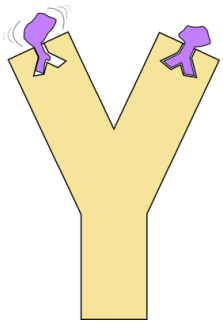
(A)



(B)

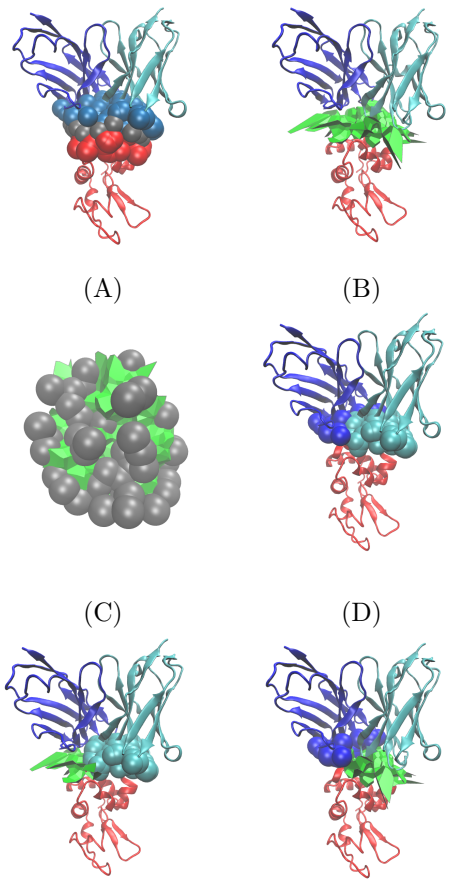


(C)



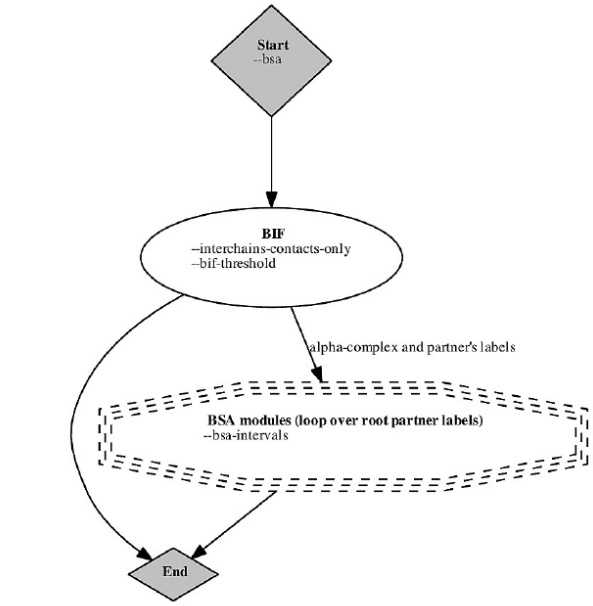
(D)

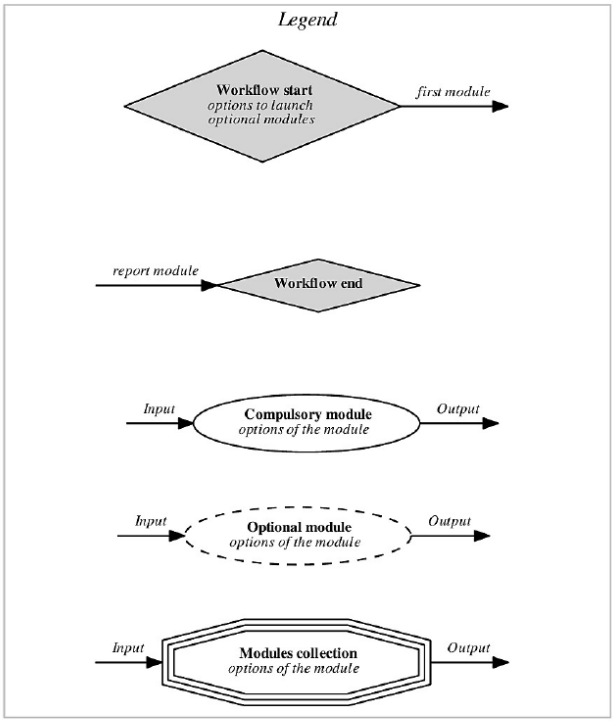
Figure 2 On the versatility of applications from the SBL: illustration on the problem of modeling interfaces, `http://sbl.inria.fr/doc/Space_filling_model_interface-user-manual.html`**.** An application is a set of programs characterized by inputs and outputs obeying the same semantics – modeling the interface between subunits of a complex in this example. From the applied standpoint, an interface is qualified by pairs of atom in contacts, interfacial solvent molecules, binding patches (*i.e.* connected components) together with their surface area and curvature, etc. Formally, an interface is parameterized by properties of the partners (defined by atoms or pseudo-atoms), as well as specific properties of these partners (*i.e.* the decomposition of a partner into groups of atoms). This parameterization allows investigating a variety of systems in a coherent way, by instantiating the same generic C++ code with different biophysical models. In the sequel, we illustrate this possibility by studying an antibody (IG) - antigen complex at two levels of detail (PDB: 1vfb): by considering the IG as a whole, and by splitting its fragment antigen binding domain (FAB) into contributions from its heavy and light chains. **(A)** Interface atoms of the IG - Ag complex: blue (atoms of the IG), read (atoms of the antigen), gray (crystal water molecules sandwiched between the partners). NB: all atoms are displayed in their solvent accessible representation i.e. radii have been expanded by 1.4. **(B)** In green, the Voronoi interface from **(A)**, which separates the partners. **(C)** Top view of the Voronoi interface from **(B)**, with interfacial water molecules. **(D)** Interfacial atoms contributed by the heavy (blue) and light chain (cyan) of the IG. **(E)** Voronoi interface of the heavy chain and interfacial atoms from the light chain. **(F)** Voronoi interface of the light chain and interfacial atoms from the heavy chain.



(A) (B) (C) (D) (E) (F)

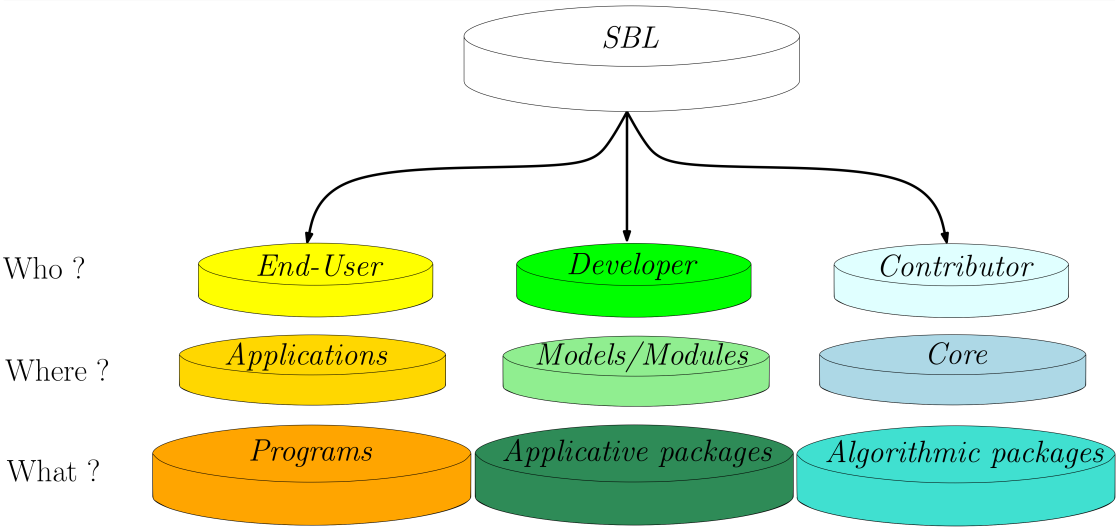
Figure 3 The structure of applications as a workflow connecting modules: workflow of the interface finder of Fig. 1(A), see `http://sbl.inria.fr/doc/Space_filling_model_interface_finder-user-manual.html`. In the `SBL`, a module is a black box transforming an input into an output. With modules, an application is a graph whose nodes are modules. By introspecting *i.e.* traversing this graph, the application generates its workflow. Moreover, nodes of the printed workflow are clickable–they point to the documentation of the software components. Practically, modules are C++ classes parameterized by classes from `SBL-CORE` and `SBL-MODELS`. This workflow indicates that there are two steps in the application considered. The first one (compulsory), reports all pairwise interfaces found in the structure. The second one (optional), may be called to computed surface areas buried on each subunit. Note that these pieces of information are summarized on the graph reported (Fig. 1(A)).





Workflow

Figure 4 Overall architecture of the `SBL`. The `SBL` is organized in focus areas for end-users, developers, and contributors. For end-users, applications solve well specified biophysical problems (Figs. 1 and 2). For developers, the `SBL` provides software components easing the development of novel applications (Fig. 3). For low level developers and contributors, the `SBL` proposes core algorithmic packages dedicated to specific processing (`CADS`: Combinatorial Algorithms and Data Structures, `GT`: Computational geometry and computational topology, `CSB`: Computational Structural Biology, `IO`: Input / Output.). The corresponding packages are interchangeable those from specific libraries, such as the BOOST C++ libraries or CGAL.



References

[1] B. Webb and A. Sali. Comparative protein structure modeling using modeller. *Current protocols in bioinformatics*, pages 5–6, 2014.

[2] A. Leaver-Fay, M. Tyka, S.M. Lewis, O.F. Lange, J. Thompson, R. Jacak, K. Kaufman, P.D. Renfrew, C.A. Smith, W. Sheffler, I.W. Davis, S. Cooper, A. Treuille, D.J. Mandell, F. Richter, Y.E. Ban, S.J. Fleishman, J.E. Corn, D.E. Kim, S. Lyskov, M. Berrondo, S. Mentzer, Z. Popovic J.J. Havranek, J. Karanicolas, R. Das, J. Meiler, T. Kortemme, J.J. Gray, B. Kuhlman, D. Baker, and P. Bradley. ROSETTA3: an object-oriented software suite for the simulation and design of macromolecules. *Methods Enzymol*, 487:545–574, 2011.

[3] B. Webb, K. Lasker, J. Velázquez-Muriel, D. Schneidman-Duhovny, R. Pellarin, M. Bonomi, C. Greenberg, B. Raveh, E. Tjioe, D. Russel, and A. Sali. Structural genomics: General applications. chapter Modeling of Proteins and Their Assemblies with the Integrative Modeling Platform, pages 277–295. Humana Press, Totowa, NJ, 2014.

[4] I. Halperin, B. Ma, H. Wolfson, and R. Nussinov. Principles of docking: An overview of search algorithms and a guide to scoring functions. *Proteins*, 47(4):409–443, 2002.

[5] B.R. Brooks, R.E. Bruccoleri, B.D. Olafson, D.J. States, S. Swaminathan, and M. Karplus. CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. *Journal of Computational Chemistry*, 4:187–217, 1983.

[6] Romelia Salomon-Ferrer, David A Case, and Ross C Walker. An overview of the Amber biomolecular simulation package. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 3(2):198–210, 2013.

[7] James C Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with namd. *Journal of computational chemistry*, 26(16):1781–1802, 2005.

[8] D. Van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. Mark, and H. Berendsen. GROMACS: fast, flexible, and free. *Journal of computational chemistry*, 26(16):1701–1718, 2005.

[9] D.J. Wales. *Energy Landscapes*. Cambridge University Press, 2003.

[10] W. Humphrey, A. Dalke, and K. Schulten. VMD: visual molecular dynamics. *Journal of molecular graphics*, 14(1):33–38, 1996.

[11] Warren L DeLano. The pymol molecular graphics system. 2002.

[12] Eric F Pettersen, Thomas D Goddard, Conrad C Huang, Gregory S Couch, Daniel M Greenblatt, Elaine C Meng, and Thomas E Ferrin. UCSF chimera–a visualization system for exploratory research and analysis. *Journal of computational chemistry*, 25(13):1605–1612, 2004.

[13] T. Hamelryck and B. Manderick. Pdb file parser and structure class implemented in python. *Bioinformatics*, 19(17):2308–2310, 2003.

[14] M. Gerstein and F.M. Richards. Protein geometry: volumes, areas, and distances. In M. G. Rossmann and E. Arnold, editors, *The international tables for crystallography (Vol F, Chap. 22)*, pages 531–539. Springer, 2001.

[15] F. Cazals, F. Proust, R. Bahadur, and J. Janin. Revisiting the Voronoi description of protein-protein interfaces. *Protein Science*, 15(9):2082–2092, 2006.

[16] F. Cazals. Revisiting the Voronoi description of protein-protein interfaces: Algorithms. In T. Dijkstra, E. Tsivtsivadze, E. Marchiori, and T. Heskes, editors, *International Conference on Pattern Recognition in Bioinformatics*, pages 419–430, Nijmegen, the Netherlands, 2010. Lecture Notes in Bioinformatics 6282.

[17] S. Loriot and F. Cazals. Modeling macro-molecular interfaces with Intervor. *Bioinformatics*, 26(7):964–965, 2010.

[18] S. Loriot, F. Cazals, M. Levitt, and J. Bernauer. A geometric knowledge-based biogeom-grained scoring potential for structure prediction evaluation. In *JOBIM*, 2009.

[19] F. Cazals, H. Kanhere, and S. Loriot. Computing the volume of union of balls: a certified algorithm. *ACM Transactions on Mathematical Software*, 38(1):1–20, 2011.

[20] F. Cazals, T. Dreyfus, S. Sachdeva, and N. Shah. Greedy geometric algorithms for collections of balls, with applications to geometric approximation and molecular coarse-graining. *Computer Graphics Forum*, 33(6):1–17, 2014.

[21] G. Bowman, V. Pande, and F. Noé. *An introduction to markov state models and their application to long timescale molecular simulation*, volume 797. Springer Science & Business Media, 2013.

[22] A. Roth, T. Dreyfus, C.H. Robert, and F. Cazals. Hybridizing rapidly growing random trees and basin hopping yields an improved exploration of energy landscapes. *Journal of Computational Chemistry*, 37(8):739–752, 2016.

[23] F. Cazals, T. Dreyfus, D. Mazauric, A. Roth, and C.H. Robert. Conformational ensembles and sampled energy landscapes: Analysis and comparison. *Journal of Computational Chemistry*, 36(16):1213–1231, 2015.

[24] J. Carr, D. Mazauric, F. Cazals, and D.J. Wales. Energy landscapes and persistent minima. *The Journal of Chemical Physics*, 144(5), 2016.

[25] D. Landau and K. Binder. *A guide to Monte Carlo simulations in statistical physics*. Cambridge university press, 2014.

[26] David S Goodsell. *The machinery of life*. Springer Science & Business Media, 2009.

[27] J.L.P. Benesch, B.T. Ruotolo, D.A. Simmons, C.V. Robinson, et al. Protein complexes in the gas phase: technology for structural genomics and proteomics. *Chemical Reviews-Columbus*, 107(8):3544–3567, 2007.

[28] D. Agarwal, C. Caillouet, D. Coudert, and F. Cazals. Unveiling contacts within macro-molecular assemblies by solving minimum weight connectivity inference problems. *Molecular and Cellular Proteomics*, 14:2274–2282, 2015.

[29] T. Dreyfus, V. Doye, and F. Cazals. Assessing the reconstruction of macro-molecular assemblies with toleranced models. *Proteins: structure, function, and bioinformatics*, 80(9):2125–2136, 2012.

[30] T. Dreyfus, V. Doye, and F. Cazals. Probing a continuum of macro-molecular assembly models with graph templates of sub-complexes. *Proteins: structure, function, and bioinformatics*, 81(11):2034–2044, 2013.

[31] F. Alber, F. Förster, D. Korkin, M. Topf, and A. Sali. Integrating diverse data for structure determination of macromolecular assemblies. *Ann. Rev. Biochem.*, 77:11.1–11.35, 2008.

[32] S. Marillet, P. Boudinot, and F. Cazals. High resolution crystal structures leverage protein binding affinity predictions. *Proteins: structure, function, and bioinformatics*, 1(84):9–20, 2015.

[33] P.L. Kastritis, I.H. Moal, H. Hwang, Z. Weng, P.A. Bates, A. Bonvin, and J. Janin. A structure-based benchmark for protein-protein binding affinity. *Protein Science*, 20:482–491, 2011.

[34] M Vihinen. No more hidden solutions in bioinformatics. *Nature*, 521(7552):261, 2015.

[35] N. Amir, D. Cohen, and H.J. Wolfson. Dockstar: a novel ilp-based integrative method for structural modeling of multimolecular protein complexes. *Bioinformatics*, 31(17):2801–2807, 2015.

[36] A. Alexandrescu. *Modern C++ design: generic programming and design patterns applied*. Addison-Wesley, 2001.

[37] H. Edelsbrunner. *Geometry and topology for mesh generation*. Cambridge, 2001.

[38] G. Shakhnarovich, T. Darrell, and P. Indyk (Eds). *Nearest-Neighbors Methods in Learning and Vision. Theory and Practice*. MIT press, 2005.

[39] H. Edelsbrunner and J. Harer. *Computational topology: an introduction*. AMS, 2010.

[40] F. Chazal, L.J. Guibas an dS.Y. Oudot, and P. Skraba. Persistence-based clustering in Riemannian manifolds. In *ACM SoCG*, pages 97–106, 2011.

[41] I. Wohlers, N. Malod-Dognin, R. Andonov, and G.W. Klau. CSA: Comprehensive comparison of pairwise protein structure alignments. *Nucleic Acids Research*, 40(W1):W303–W309, 2012.

[42] H-J. Woo and B. Roux. Calculation of absolute protein–ligand binding free energy from computer simulations. *PNAS*, 102(19):6825–6830, 2005.

[43] M.K. Gilson and H-X. Zhou. Calculation of protein-ligand binding affinities. *Annual review of biophysics and biomolecular structure*, 36(1):21, 2007.

[44] A. Vangone and A. Bonvin. Contacts-based prediction of binding affinity in protein–protein complexes. *eLife*, 4:e07454, 2015.

[45] Donald E Knuth. Computer programming as an art. In *ACM Turing award lectures*, page 1974. ACM, 2007.

6 Supplemental

In section 6.1, we show how the development of a complex applications stems directly from the connexion of modules. As an example, we provide the code for a simplified version of our application computing surfaces and volumes (details at `http://sbl.inria.fr/doc/Space_filling_model_surface_volume-user-manual.html`). While this simplified version computes the volume of a union of geometric balls without any specific meaning, the version from the `SBL` provides switches to handle this case and also that of atomic models, with complete control of the PDB format and its features (filtering hydrogen atoms, handling duplicates, filtering on occupancy factors, etc).

In section 6.2, we show (i) how to use the batch manager to run multiple calculations (details at `http://sbl.inria.fr/doc/Batch_manager-user-manual.html`), and (ii) how to easily extract statistics using the application *Python Analysis L Scale Experiments* (PALSE, details at `http://sbl.inria.fr/doc/PALSE-user-manual.html`).

6.1 Simplified version of `sbl-vorlume-txt.exe`

sbl-vorlume-example.cpp The following piece of code defines a program taking as input a file listing 3D balls—each line describes one ball by its center and radius, and computes the volume and the surface of their union.

```
//Loading the input file listing the 3D spheres.
#include <SBL/Models/Spheres_3_file_loader.hpp>

//Modules required for computing the volume of the union of balls.
#include <SBL/Modules/Alpha_complex_of_molecular_model_module.hpp>
#include <SBL/Modules/Union_of_balls_surface_volume_3_module.hpp>

//Workflow
#include <SBL/Modules/Module_based_workflow.hpp>

//Models to be used for defining the context of the application.
#include <SBL/Models/Geometric_particle_traits.hpp>
#include <SBL/CSB/Alpha_complex_of_molecular_model.hpp>
//Alpha-complex module requirements for dumping VMD and PyMOL
//visualization files.
#include <SBL/IO/Alpha_complex_of_molecular_model_molecular_view.hpp>

//Traits class defining the types used by the modules
struct Traits
{
  typedef SBL::Models::T_Geometric_particle_traits<>   Particle_traits;
  typedef std::vector<Particle_traits::Particle_type>  Particles_container;
  typedef SBL::CSB::T_Alpha_complex_of_molecular_model
  <Particle_traits>                                    Alpha_complex;
};

//Definition of the loaders and modules to be used.
typedef SBL::Models::T_Spheres_3_file_loader
<Traits::Particle_traits::Particle_type>                Loader;
typedef SBL::Modules::T_Alpha_complex_of_molecular_model_module
<Traits>                                                Alpha_complex_module;
typedef SBL::Modules::T_Union_of_balls_surface_volume_3_module
<Traits>                                                Surface_volume_module;
```

```
//Updating the alpha complex of the volume module from the previous module
void set_alpha_complex(Alpha_complex_module* alpha_complex_module,
                       Surface_volume_module* surface_volume_module)
{surface_volume_module->get_alpha_complex() =
  &alpha_complex_module->get_alpha_complex();}

int main(int argc, char *argv[]){

  //Building the workflow
  SBL::Modules::Module_based_workflow workflow("sbl-vorlume-example");

  //-- adding the loader
  Loader* loader = workflow.add_loader<Loader>("Spheres Loader");

  //-- adding the first module (alpha-complex)
  SBL::Modules::Module_based_workflow::Vertex
  u = workflow.register_module<Alpha_complex_module>("Alpha Complex");
  workflow.set_start_module(u);

  //-- adding the second and last module (surface / volume)
  SBL::Modules::Module_based_workflow::Vertex
  v = workflow.register_module<Surface_volume_module>("Surface / Volume");
  workflow.set_end_module(v);

  //-- connecting the latter two modules, specifying how to update
  //the input of the second module from the output of the first module.
  workflow.make_module_flow(u, v, &set_alpha_complex, "alpha-complex");

  //Read the command line and initialize the module's parameters.
  workflow.parse_command_line(argc, argv);

  //Load the data and initialize the alpha complex with the input 3D spheres
  workflow.load();
  ((Alpha_complex_module&)workflow.get_module(u)).get_particles() =
  &loader->get_geometric_model();

  //Start the workflow
  workflow.start();

  return 0;
}
```

CMakeLists.txt The program sbl-vorlume-example.cpp is compiled using CMake. The following code defines the corresponding CMakeLists.txt file.

```
cmake_minimum_required(VERSION 2.6)

set(SBL_USE_LIBS CGAL MPFR GMP Boost)
find_package(SBL)
include(${SBL_USE_FILE})
create_sbl_program(sbl-vorlume-example)
```

Running the executable. Assuming that the SBL library has been installed in a standard location, the following command lines respectively compile and execute the program :

```
> mkdir build; cd build;
> cmake .. ; make;
> ./sbl-vorlume-example.exe
```

Note that if the library is not installed in a standard location, CMake has to know where is located the library, as specified in the documentation `http://sbl.inria.fr/doc/sbl-devel-with-sbl-tutorial.html`. This can be achieved in several ways, depending on your configuration :

- if the library is not installed at all :

  ```
  > cmake .. -DSBL\_DIR=</path/to/sbl/dir>
  ```

- if the library is installed in a non-standard location :

  ```
  > cmake .. -DCMAKE\_MODULE\_PATH=</path/to/sbl/dir>/share/cmake/Modules
  ```

6.2 Managing batches of executions, and using PALSE to extract statistics

example-batch-palse-vorlume.py The following Python script reads an input list of directories containing text files of 3D spheres, run the vorlume example over all the files in each directory, and compute the mean volume for each directory. It uses the package Batch_manager for making and runing batches of calculations, and the package PALSE for running analysis over the batches.

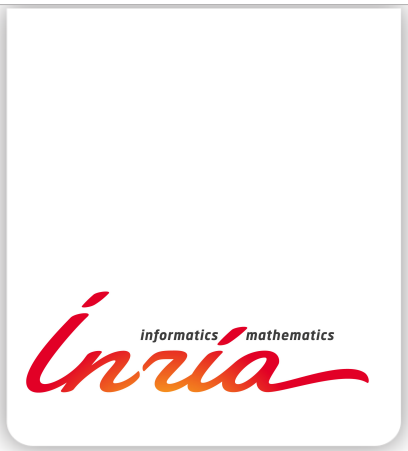
```
#! /usr/bin/python

from SBL import Batch_manager
from Batch_manager import *
from SBL import PALSE
from PALSE import *

for i in range(1, len(sys.argv)):

    batch = BM_Batch()
    #Load files of the dataset.
    batch.load_dataset(sys.argv[i])
    #Load the batch-vorlume-txt.spec specification to compute the volume
    # of the balls in the file.
    batch.load_run_specification("batch-vorlume-txt.spec")
    #Run.
    batch.run()

    database = PALSE_xml_DB()
    database.load_from_directory(
    "sbl-vorlume-example-"+os.path.basename(sys.argv[i]),".*volumes.xml")

    volumes = database.get_all_data_values_from_database(
    "restrictions/item/total_volume", float)
    volumes = PALSE_DS_manipulator.convert_listoflists_to_list(volumes)
    print(sum(volumes)/float(len(volumes)))
```

batch-vorlume-txt.spec The following is the specification file used by the Batch_manager package from the Python script example-batch-palse-vorlume.py . Please note that the second line of the specification file corresponds to the path to the program to be executed : the program should be in one's $PATH – or the line modified to set the path to the executable.

```
#The executable.
EXECUTABLE sbl-vorlume-example.exe

#Any .txt file from the dataset is elibigle for the option -f,
# as specified by the python regex.
#NB: the dataset name is specified when constructing the batch in the python script.
IFO f "\.txt$"

#There is a unique IFO, with one execution per value i.e. file.
IFO-ASSOC-RULE unary

#These are the Non File Options for the executable.
NFO log
NFO output-prefix
NFO verbose
```

Running the executable. The following command line executes the Python script :

```
> chmod u+x example-batch-palse-vorlume.py
> ./example-batch-palse-vorlume.py <path/to/input/directory>
```



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399